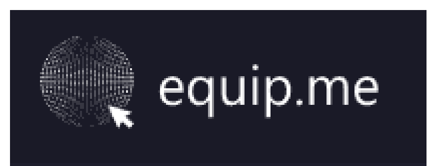

Модульный Laravel,
или Как собрать фичу в кучу


php
PHP Russia
2022


Алексей Васильев
Equip Group

О докладчике



- Fullstack-разработчик в компании **Equip Group** (оптовая торговля оборудованием)
- Разрабатываю проекты на Laravel более 3 лет
- Интересуюсь структурированием кода в разных аспектах



О чём пойдёт речь

- Типовая структура Laravel-проекта и её проблемы
- Пакет, собирающий фичу воедино
- Как это работает
- От структурных модулей к функциональным
- Преимущества и недостатки
- Сравнение с Lucid
- Заключение

Проблемы типовой структуры Laravel-проекта



Что такое фича

- Сравнительно обособленная единица функционала (например, раздел или подсистема)
- Структурные компоненты фичи: классы, контроллеры, маршруты, шаблоны, наборы настроек



Структура проекта

- app
- bootstrap
- config
- database
- public
- resources
- routes
- storage
- tests
- vendor
- .editorconfig
- .env
- .env.example
- .gitattributes
- .gitignore
- .styleci.yml
- artisan
- composer.json

→

- Console
- Exceptions
- Helpers
- Http
  - Controllers
  - Middleware
  - Requests
  - Kernel.php
- Models
- Providers

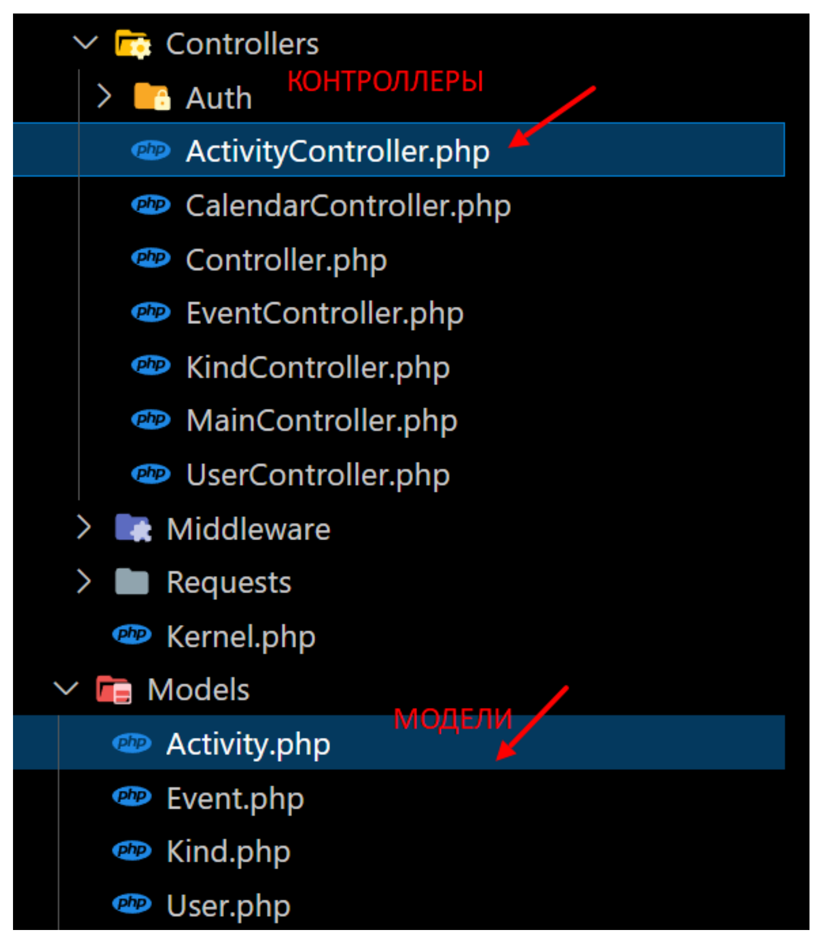


Структура проекта

- Файлы разделены по типам: контроллеры, request'ы, маршруты, шаблоны, конфигурация



Что со структурой не так



resources
js
lang
sass
views
ШАБЛОНЫ
routes
api.php
channels.php
console.php
web.php
МАРШРУТЫ



Что со структурой не так

- При поддержке и развитии фичи так или иначе затрагивается весь проект, есть риск что-то зацепить
- С выделением фичи в отдельный сервис ещё сложнее



Пакет, собирающий фичу воедино

garmonic/laravel-features-arch



Что легко и что сложно переместить

Легко

- Контроллеры
- Прочие классы

Сложно

- Шаблоны
- Маршруты
- Конфигурация



Загрузка маршрутов

```php
/* RouteServiceProvider::boot() */
$featureName = $this->feature->getName();
$routeBase = $this->feature->getRouteBase();

$this->routes(function () use ($featureName, $routeBase) {
    Route::prefix('api/'.$routeBase)
        ->name('api.'.$featureName.'.')
        ->middleware('api')
        ->group($this->getBasePath('routes/api.php'));
```




Загрузка маршрутов

```php
protected function getBasePath(string $path = '')
{
    $featureName = $this->feature->getName();
    $mainPath = 'features/'.$featureName;
    if ($path !== '') {
        $mainPath .= '/'.$path;
    }
    return base_path(path: $mainPath);
}
```




Загрузка маршрутов

- Свой **`RouteServiceProvider`** на основе Laravel'овского
- Загружает файлы маршрутов, исходя из пути к директории фичи



Загрузка шаблонов

```php
/* FeatureServiceProvider::register() */
$featuresList = config('features.list', []);
foreach ($featuresList as $alias => $feature) {
    App::singleton($alias, function () use ($feature) {
        return new $feature();
    });
    $pathToViews = base_path('features/'.$alias.
                              '/resources/views');
    View::addNamespace($alias, $pathToViews);
}
```




Загрузка шаблонов

- Решается через добавление неймспейсов по именам фич
- Реально это выполняет основной сервис-провайдер пакета — **`FeatureServiceProvider`**



Загрузка конфигурации

```
/* Feature::loadConfig() */
$this→configuration = new Repository();
$configPath = realpath(app()→basePath().'/features/'.
                        $this→name.'/config');

foreach (Finder::create()
            →files()
            →name('*.php')
            →in($configPath) as $file) {
    $directory = $this→getNestedDirectory($file, $configPath);

    $files[$directory.basename(
        $file→getRealPath(),
        '.php'
    )] = $file→getRealPath();
}
```

Загрузка конфигурации

- Сделана по аналогии с загрузкой конфигурации всего проекта, только с заменой директории для поиска настроечных файлов



Базовый класс

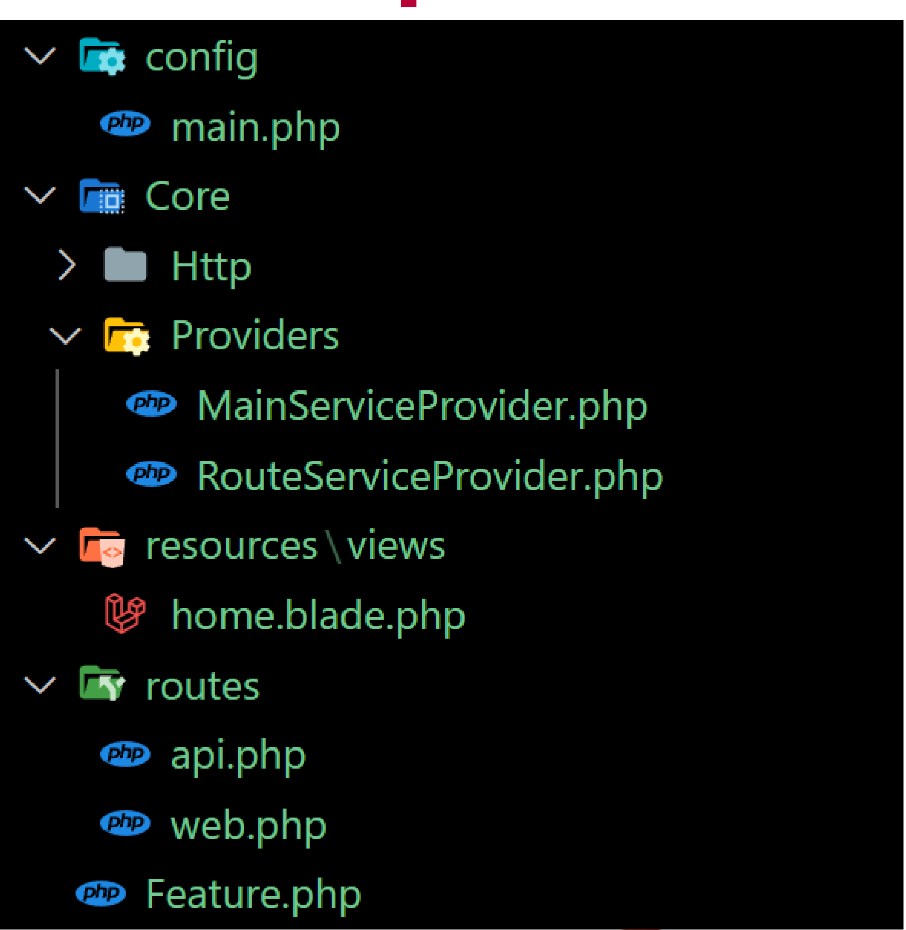
- Загружает конфигурацию
- Загружает контейнер
- Загружает сервис-провайдеры
- «Точка входа»: через него получаются маршруты, шаблоны, настройки
- Доступен в контроллерах как свойство **`feature`**, пробрасывается в шаблоны как переменная **`$feature`**

Как это работает



Создание заготовки фичи

```
$ php artisan make:feature Profile
Creating feature Profile
Route base is /profile
Feature created successfully
```





Создание заготовки фичи

- Консольная команда **`php artisan make:feature`**
- Из стабов создаются сервис-провайдеры, пустые файлы конфигурации, маршрутов, демо-шаблон



Загрузка фич

```php
/* config/features.php */
'list' => [
    'Catalog' => Features\Catalog\Feature::class,
    'News' => Features\News\Feature::class
]
```




Загрузка фич

```php
/* FeatureServiceProvider::register() */
$featuresList = config('features.list', []);
foreach ($featuresList as $alias => $feature) {
    App::singleton($alias, function () use ($feature) {
        return new $feature();
    });
    $pathToViews = base_path('features/'.$alias.
                            '/resources/views');
    View::addNamespace($alias, $pathToViews);
}
```

ПОЛУЧЕНИЕ ИЗ КОНФИГУРАЦИИ

ПРИВЯЗКА В КОНТЕЙНЕР



Загрузка фич

- Загружаются только фичи, включённые в общей конфигурации
- `FeatureServiceProvider` привязывает базовые классы по алиасам в общий контейнер приложения



Относительный роутинг

- Файлы маршрутов сохраняются в поддиректорию `/routes` директории фичи
- Ссылки на соответствующие маршруты получаются через обращения к методу `route()` базового класса
- Пример: `return redirect($this->feature->route('home'))`



Внутренняя конфигурация

- Файлы конфигурации сохраняются в поддиректорию **`/config`** директории фичи
- Ссылки на соответствующие ключи получаются через обращения к методу **`config()`** базового класса
- `@if($feature->config('main.showButton')) <button> @endif`



Шаблоны фичи

- Файлы шаблонов сохраняются в поддиректорию **`/resources/views`** директории фичи
- Отдача шаблонов происходит через метод **`view()`** базового класса
- `return $this->feature->view('list', ['items'=>$items]);`



Контейнер фичи

- Фича имеет свой собственный сервис-контейнер, работающий аналогично контейнеру приложения
- Привязка в контейнер выполняется через **`$feature->bind()`** или **`$feature->singleton()`**, получение — через **`$feature->make()`**



От структурных модулей к функциональным



Фича как функциональный модуль

- Следующий шаг — превращение фичи из структурного модуля в функциональный
- API для внешних обращений оформляется как методы базового класса
- Все обращения к фиче снаружи должны идти через эти методы
- Это позволяет отвязать внутренности модуля от внешних классов и при необходимости вынести его в отдельный сервис



Фича как функциональный модуль

```php
/* Features\Catalog\Feature */
public function getSectionById(int $id, $onlyActive = true)
{
    return Section::when($onlyActive, function ($query) {
        $query->active();
    })->where('id', $id)->first();
}
```




Фича как функциональный модуль

```php
/* Features\Catalog\Feature */
public function getProductById(int $id, $onlyActive = true)
{
    return Product::when($onlyActive, function ($query) {
        $query->active();
    })->where('id', $id)->first();
}
```




Фича как функциональный модуль

```php
/* Features\News\Core\Http\Controllers\MainController  */
$catalog = app('Catalog');
$product = null;
if ($catalog instanceof Catalog && !is_null($publication->product_id)) {
    $product = $catalog->getProductById($publication->product_id);
    $product->url = $catalog->route('product',
                                   ['product'=>$product->id]);
}
```




Преимущества и недостатки



Преимущества

- Пакет решает задачу вынесения фичи в компактный фрагмент структуры проекта
- Даёт движок, загружающий правильно расположенные файлы + инструмент генерации первоначальной структуры
- Даёт возможность делать фичи модулями не только структурно, но и функционально
- Может внедряться постепенно, только для новых фич



Недостатки

- Фичи привязываются в контейнер по именам и теоретически могут конфликтовать с другими классами
- Некоторая «сырость» пакета, в том числе отсутствие подробной документации, генераторов отдельных файлов



FeaturesArch vs. Lucid



Сходства

- Собирают в одной директории файлы одной фичи
- Представляют фичу (в Lucid — сервис) как структурный и программный модуль
- Позволяют включать и выключать фичи



Различия

	FeaturesArch	Lucid
Обработка запроса	Как в Laravel	Controller → Feature → Job
Генераторы	Только начальная структура	Начальная структура и отдельные компоненты
Воздействие на структуру проекта	Директория `/features` и свой конфиг-файл	Затрагивает всю структуру, в том числе директорию `/app`
Для чего выбирать	Для структурирования новых или имеющихся проектов с сохранением общего подхода	Для новых проектов с прицелом на лучшие практики



Заключение



Как установить пакет

- `composer require garmonic/laravel-features-arch`
- `php artisan vendor:publish --provider='Garmonic\FeaturesArch\Providers\FeatureServiceProvider'`
- Добавить в **`composer.json`** (секция «autoload.psr4») строку: **`"Features\\": "features/"`**



Контакты

- **E-mail:** garmonic@protonmail.com
- **Telegram:** @Alex_lotus
- **Канал:** https://t.me/code_way
- **Слайды:** http://bit.ly/3WRIAYd



Оценить доклад

